Distributed Storage

Distributed Systems Spring 2019

Lecture 22

Plan For Remaining Lectures

- Distributed File Systems
- Distributed Machine Learning
- Distributed Resource Allocation

NFS: Networked File System

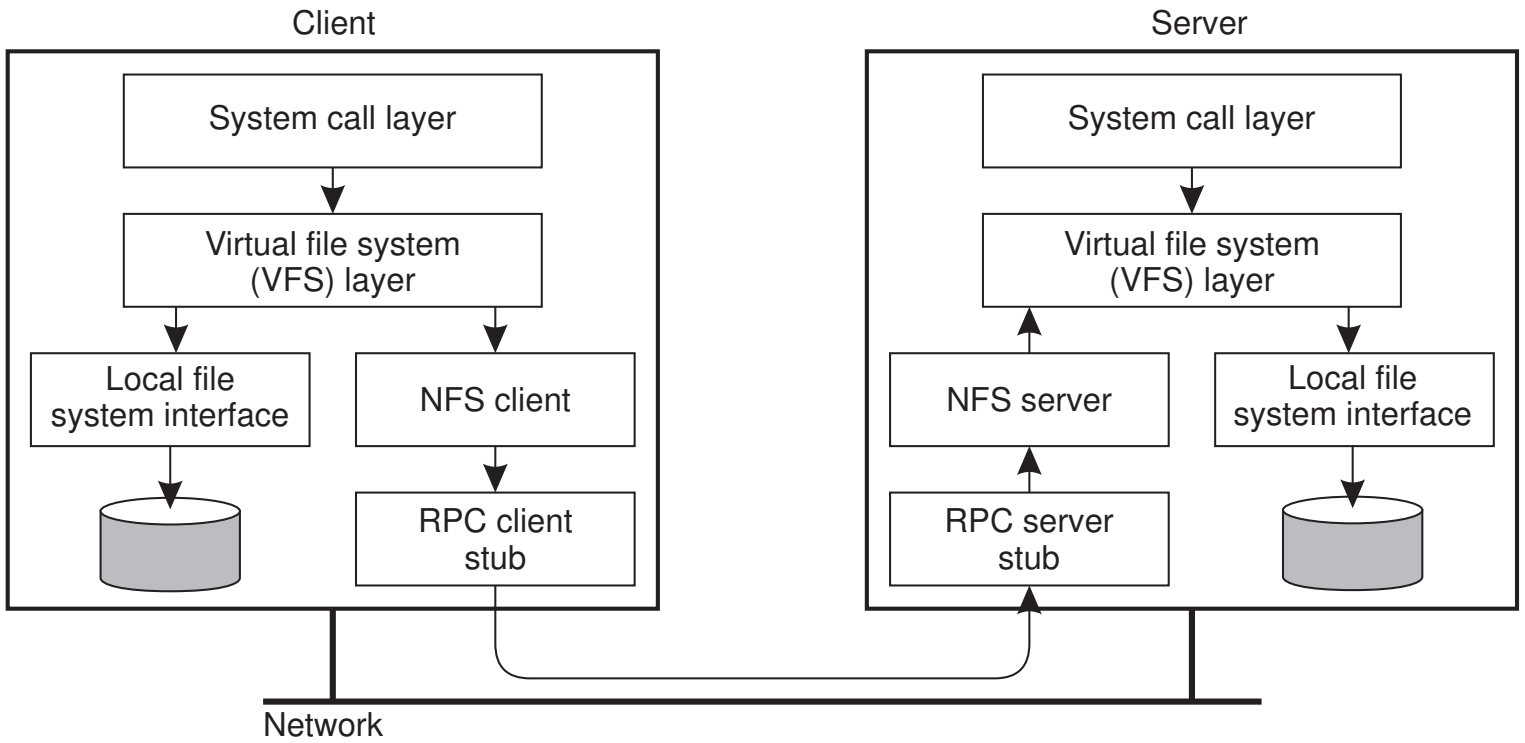
- SUN in 1984
- First use-case of SunRPC
- Abstraction of a UNIX/POSIX FS
- Popular use-case: centralized home directories

NFS Goals: Make operations appear:

1. Local
2. Consistent
3. Fast

NFS Design and Implementation

- Stateless protocol: clients specify exact state in requests (read(offset))
- TCP or UDP
- Clients and server use RPC's



NFS Operation

Typical program:

1. fd = open("path", flags)
2. read(fd, buf, n)
3. write(fd, buf, n)
4. close(fd)

- Server maintains fd to inode,offset mapping
- Clients can cache file data and metadata to reduce network transfers
- Write-behind: writes sent on close or fsync

Race condition

- Two clients
- First does open("dir1/f") followed by read(fd)
- Second client: rename("dir1", "dir2"); rename("dir3","dir1");

Which file should be read?

- dir1/f or dir2/f ?
- Conventional FS in UNIX: dir2/f (linearizable)

Concurrent Writes:

- Clients can issue concurrent operations on a file
- NFS enforces nothing!

File Handles

- Another layer of indirection
- fh = lookup("path")
- File handles are opaque identifiers provided to a client
- Includes info needed to identify file on server
- fh = (volume ID, inode #, generation #,..)
- Trick: Store server state at the client
- Operations are thus idempotent
- Generation numbers similar to Multi version concurrency control
- Client 1: unlink("f"); fd = open("f", creat)
- Client 2 : fd=open("f",readonly) ; read(fd)
- Client 2 can still continue to interact with "correct" file

Failures

Server fails (crashes):

- In-memory data on server is lost

Client crashes:

- Lose data in client cache

Lost messages:

- Retry operation
- Delete file twice?

Consistency

- NFS provides "close-to-open" consistency
- Clients should always ask server for updates before open()
- No guarantees for multiple writes

Alternative: Poll server on every open

- Heavy load on server
- Round trip time

Alternate techniques:

- Server callbacks (used in AFS)
  - Server must keep state about files and clients
  - What if a client disappears?
- Leases

Distributing Storage

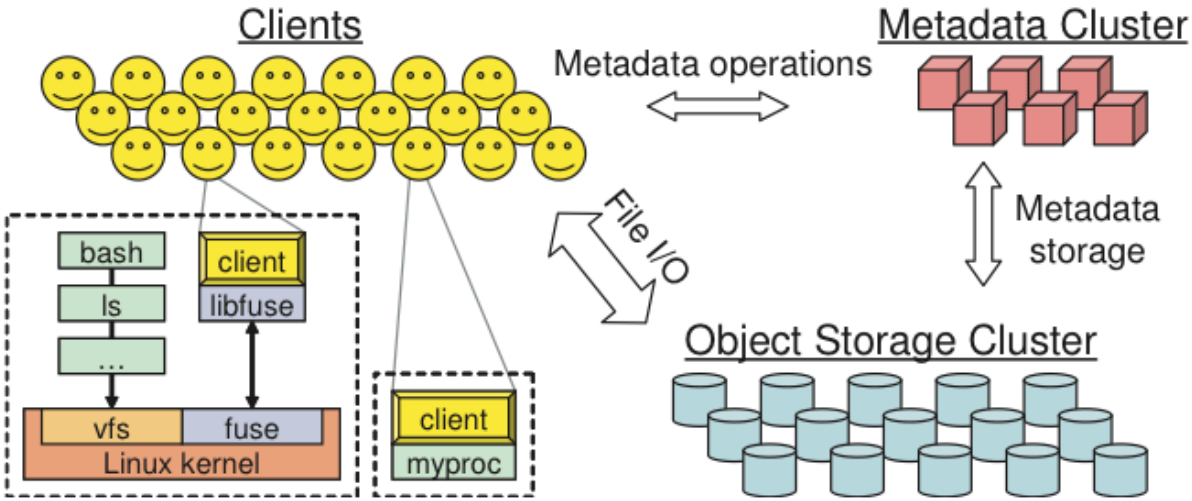
- NFS stores data in one FS/server
- For large data sizes and I/O parallelism, want to store on multiple servers
- NFS performance can be a huge issue
  - Handling concurrent writes is hard
  - Lots of small files in home directories...

The Metadata Storage Problem

- Metadata: Map file path to location
- GFS: Stores metadata on separate server in memory
- Ok for chunk-based file systems
- Metadata overhead can be quite large in general
- How to distribute metadata?
- Many challenges: large number of files, clients, servers
- All these are dynamically changing (esp servers)

Ceph

- Truly decentralized: data can reside on multiple machines
- Object: Key-val store
- Block: VM virtual disk
- POSIX-like File Systems



CRUSH

- Controlled Replication Under Scalable Hashing
- Similar to consistent hashing
- Object Name to placement group to list of OSD's
- Two steps:
  1. Hash(object name) to placement group
  2. CRUSH(placement group, cluster topology) to list of OSD's

CRUSH properties

- No need to store object locations explicitly
- Stable: Very little data movement when topology changes
- Reliable: placement constrained by failure domains (racks etc)
- Flexible: replication, erasure codes, complex placement schemes...

Metadata Storage

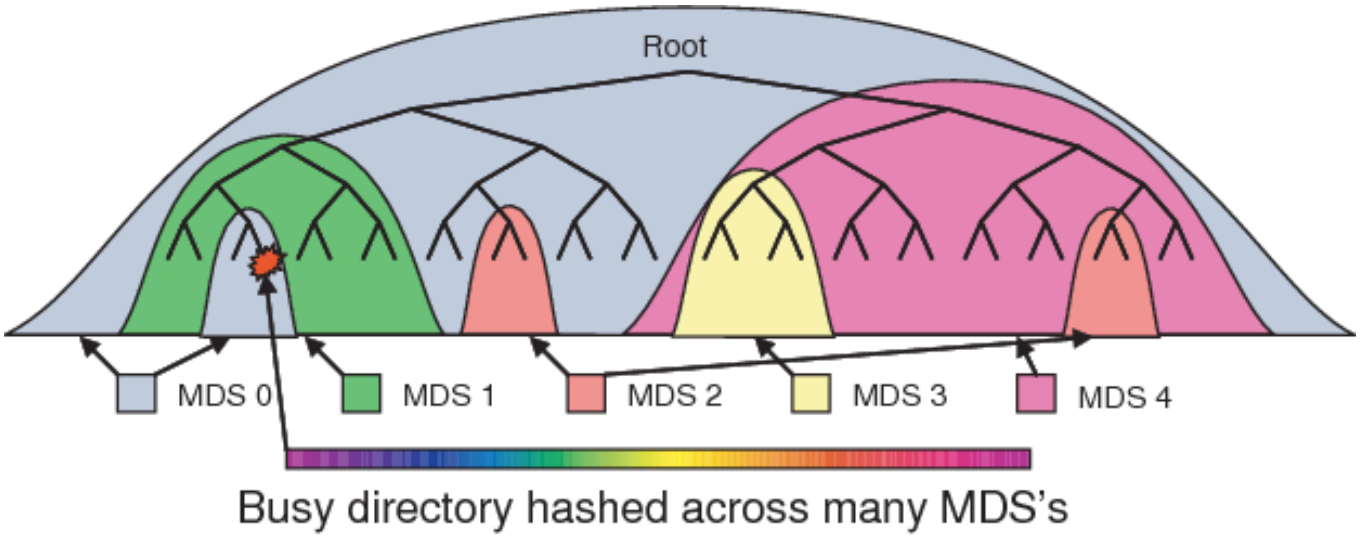
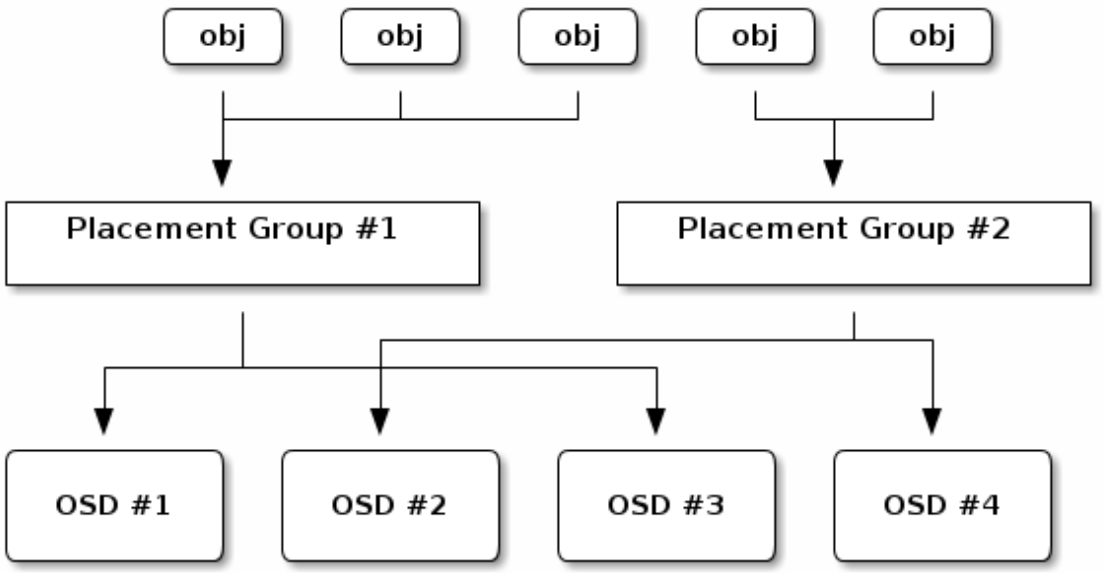
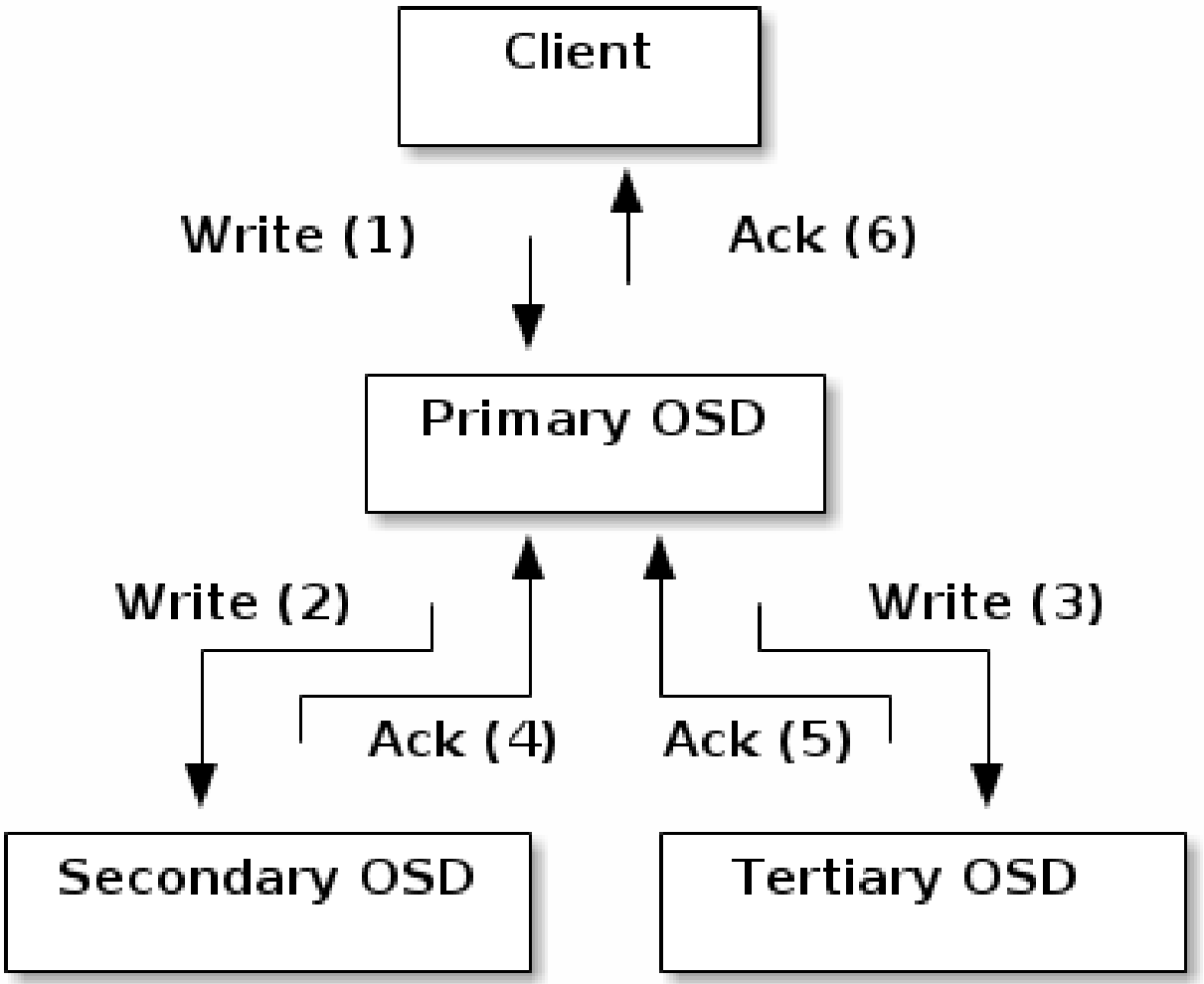


Figure 2: Ceph dynamically maps subtrees of the direc

Ceph 2-level Hashing



Ceph Writes


Client
Write (1)
Ack (6)
Primary OSD
Write (2)
Write (3)
Ack (4)
Ack (5)
Secondary OSD
Tertiary OSD

Erasure Codes